

---

# **openFlightHPC-docs Documentation**

**openFlightHPC**

**Jan 26, 2021**



---

## Deploying Resources

---

<b>1</b>	<b>Documentation Goal</b>	<b>3</b>
<b>2</b>	<b>Acknowledgements</b>	<b>5</b>
<b>3</b>	<b>Table of Contents</b>	<b>7</b>



This site contains the documentation for the OpenFlightHPC project. It contains tips and tools for streamlining the building and management of a HPC research environments. While the documentation will mainly focus on the OpenFlightHPC Research Environment workflow there will be notes and tips to customising the process for varying workflows.



# CHAPTER 1

---

## Documentation Goal

---

The purpose of this documentation is to provide understandable guidance to delivering the OpenFlightHPC Research Environment. This includes deploying resources on a given platform and installing the relevant software.

*Environment Delivery* is the installation of software for the user experience on the research environment. This usually involves some sort of resource management/queuing system and application installation.

---

**Note:** It is recommended to read through all of the documentation before starting to design the HPC platform to understand the scope and considerations.

---



## CHAPTER 2

---

### Acknowledgements

---

We recognise and respect the trademarks of all third-party providers referenced in this documentation. Please see the respective EULAs for software packages used in configuring your own environment based on this knowledgebase.

### 2.1 License

This documentation is released under the [Creative-Commons: Attribution-ShareAlike 4.0 International](#) license.



### 3.1 Workflow: AWS

#### 3.1.1 Overview

This workflow describes deploying cloud resources, consisting of:

- A 'domain' containing a network and security group
- A gateway node with internet access
- 2 to 8 compute nodes with internet access

#### 3.1.2 Prerequisites

This document presumes the following situation:

- The appropriate cloud tool is installed and configured with access details
- There is enough availability in the upstream cloud region of your account to deploy these resources
- A suitable CentOS 7/8 source image is available in the cloud provider for basing nodes off of
  - This source image should be at least 16GB in size to allow enough space for applications and data

---

**Note:** OpenFlight provides various images that are ready for cloud, these can be found at <https://openflighthpc.org/images/>

---

#### 3.1.3 Template Parameters

There are multiple parameters in the template, some are required and others are optional, these are outlined below:

- `sourceimage` - The AMI ID to use for both the gateway and compute nodes

- `clustername` - The name of the cluster, to be used as part of the FQDN for nodes
- `customdata` - A base64 encoded cloudinit string for both the gateway and compute nodes
- `computeNodesCount` - The number of compute nodes to deploy, a value between 2 and 8 (default: 2)
- `gatewayinstancetype` - The instance type to be used for the gateway node (default: t2.small)
- `computeinstancetype` - The instance type to be used for the compute nodes (default: t2.small)

### 3.1.4 Deploy Resources

- Download the OpenFlight AWS template:

```
$ curl -o cluster.yaml https://raw.githubusercontent.com/openflighthpc/openflight-compute-cluster-builder/master/templates/aws/cluster.yaml
```

- Generate base64 `customdata` string to set the default username to `flight` and add an SSH public key for authentication:

```
$ DATA=$(cat << EOF
#cloud-config
system_info:
  default_user:
    name: flight
runcmd:
  - echo "ssh-rsa MySSHpublicKeyHere user@host" >> /home/flight/.ssh/authorized_
  ↪keys
EOF
)
$ echo "$DATA" |base64 -w 0
I2Nsb3VkLWNvbmZpZwpzeXN0ZW1faW5mbWogIGRlZmF1bHRfdXNlcjoKICAgIG5hbWU6IGZsaWdodApydW5jbWQ6CiAgLSB1
```

---

**Note:** The base64 value will differ from the above depending on the SSH key specified. It does *not* need to match the example output above.

---

- Deploy a cluster (with a gateway and 2 nodes):

```
$ aws cloudformation deploy --template-file cluster.yaml --stack-name mycluster \
--parameter-overrides sourceimage="AMI_ID_HERE" \
clustername="mycluster" \
customdata=
↪"I2Nsb3VkLWNvbmZpZwpzeXN0ZW1faW5mbWogIGRlZmF1bHRfdXNlcjoKICAgIG5hbWU6IGZsaWdodApydW5jbWQ6CiAgLSB1
↪"
```

---

**Note:** The above command can be modified to override the other parameters mentioned at the beginning of this page. The 3 parameters used in the above command are the minimum required to bring a cluster up

---

- Configure `/etc/hosts` on gateway and nodes:

```
$ cat << EOF >> /etc/hosts
10.10.3.67    gateway1
10.10.8.10   node01
10.10.4.54   node02
EOF
```

**Note:** The IP addresses of your nodes may differ. Use the CLI tools or GUI to determine the internal IP addresses to be used in the hosts file.

---

- Setup passwordless root SSH to all compute nodes from the gateway. This can be done by generating a public key with `ssh-keygen` and adding it to `/root/.ssh/authorized_keys` on the gateway and all other nodes.

## 3.2 Workflow: Azure

### 3.2.1 Overview

This workflow describes deploying cloud resources, consisting of:

- A 'domain' containing a network and security group
- A gateway node with internet access
- 2 to 8 compute nodes with internet access

### 3.2.2 Prerequisites

This document presumes the following situation:

- The appropriate cloud tool is installed and configured with access details
- There is enough availability in the upstream cloud region of your account to deploy these resources
- A suitable CentOS 7/8 source image is available in the cloud provider for basing nodes off of
  - This source image should be at least 16GB in size to allow enough space for applications and data

---

**Note:** OpenFlight provides various images that are ready for cloud, these can be found at <https://openflighthpc.org/images/>

---

### 3.2.3 Template Parameters

There are multiple parameters in the template, some are required and others are optional, these are outlined below:

- `sourceimage` - The AMI ID to use for both the gateway and compute nodes
- `clustername` - The name of the cluster, to be used as part of the FQDN for nodes
- `customdata` - A base64 encoded cloudinit string for both the gateway and compute nodes
- `computeNodesCount` - The number of compute nodes to deploy, a value between 2 and 8 (default: 2)
- `gatewayinstancetype` - The instance type to be used for the gateway node (default: Standard DS1 v2)
- `computeinstancetype` - The instance type to be used for the compute nodes (default: Standard DS1 v2)

### 3.2.4 Deploy Resources

- Download the OpenFlight Azure template:

```
$ curl -o cluster.json https://raw.githubusercontent.com/openflighthpc/openflight-  
compute-cluster-builder/master/templates/azure/cluster.json
```

- Generate base64 customdata string to set the default username to `flight` and add an SSH public key for authentication:

```
$ DATA=$(cat << EOF  
#cloud-config  
system_info:  
  default_user:  
    name: flight  
runcmd:  
  - echo "ssh-rsa MySSHpublicKeyHere user@host" >> /home/flight/.ssh/authorized_  
keys  
EOF  
)  
$ echo "$DATA" |base64 -w 0  
I2Nsb3VklWNvbmZpZwpzeXN0ZW1faW5mbwogIGRlZmF1bHRfdXNlcjoKICAgIG5hbWU6IGZsaWdodApydW5jbWQ6CiAgLSB1
```

---

**Note:** The base64 value will differ from the above depending on the SSH key specified. It does *not* need to match the example output above.

---

- Create resource group for the cluster:

```
$ az group create --name mycluster --location "UK South"
```

- Deploy a cluster (with a gateway and 2 nodes):

```
$ az group deployment create --name mycluster --resource-group mycluster \  
--template-file cluster.json \  
--parameters sourceimage="SOURCE_IMAGE_PATH_HERE" \  
clustername="mycluster" \  
customdata=  
"I2Nsb3VklWNvbmZpZwpzeXN0ZW1faW5mbwogIGRlZmF1bHRfdXNlcjoKICAgIG5hbWU6IGZsaWdodApydW5jbWQ6CiAgLSB1  
"
```

---

**Note:** The above command can be modified to override the other parameters mentioned at the beginning of this page. The 3 parameters used in the above command are the minimum required to bring a cluster up

---

- Setup passwordless root SSH to all compute nodes from the gateway. This can be done by generating a public key with `ssh-keygen` and adding it to `/root/.ssh/authorized_keys` on the gateway and all other nodes.

## 3.3 Workflow: Ansible

### 3.3.1 Overview

This workflow describes configuring a simple HPC environment, consisting of:

- Shared NFS directories for users, data and applications
- SLURM queuing system for workload processing and management
- Flight Env for managing configuration and applications available in the environment

### 3.3.2 Prerequisites

This document presumes the following situation:

- The cluster has a gateway node (for running various servers)
- The cluster has multiple compute nodes (for executing jobs)
- DNS is correctly configured to allow hostname connections between the nodes
- Firewall connections between the gateway and compute nodes are open to allow various services to communicate (e.g. queuing system, nfs, etc)
- SSH keys are correctly configured to allow the gateway to login to nodes (as root)
- There is sufficient storage space on the gateway and compute nodes (for applications and data, recommended 16GB+)

### 3.3.3 Configure Environment

- Install ansible (>v2.8.0):

```
$ yum install -y epel-release
$ yum install -y ansible
```

- Create hosts file:

```
$ cat << EOF > /etc/ansible/hosts
[gateway]
gateway1

[compute]
node01
node02
EOF
```

- Setup playbook:

```
$ yum install -y git
$ git clone https://github.com/openflighthpc/openflight-ansible-playbook
```

**Warning:** It is highly recommended to inspect all roles and edit them to your requirement or, alternatively, write your own roles. These roles are provided “as is” and no guarantee is made that the roles will function properly in environments different to that of the example environment used in this documentation.

- Run playbook:

```
$ cd openflight-ansible-playbook
$ ansible-playbook openflight.yml
```

---

**Note:** The playbook may hang trying to verify the SSH fingerprints of the hosts if none of them have been logged into before from the ansible host. It is recommended to have already established a trusted SSH connection to all systems first.

---

## 3.4 Workflow: Manual

### 3.4.1 Overview

This workflow describes configuring a simple HPC environment, consisting of:

- Shared NFS directories for users, data and applications
- SLURM queuing system for workload processing and management
- Flight Env for managing configuration and applications available in the environment

### 3.4.2 Prerequisites

This document presumes the following situation:

- The cluster has a gateway node (for running various servers)
- The cluster has multiple compute nodes (for executing jobs)
- DNS is correctly configured to allow hostname connections between the nodes
- Firewall connections between the gateway and compute nodes are open to allow various services to communicate (e.g. queuing system, nfs, etc)
- SSH keys are correctly configured to allow the gateway to login to nodes (as root)
- There is sufficient storage space on the gateway and compute nodes (for applications and data, recommended 16GB+)

### 3.4.3 NFS or Other Shared Storage

NFS is not required for the shared storage solution, however, some form of shared storage is highly recommended to ensure that all nodes in the research environment can access applications and data.

Information on NFS configuration is available in the [official NFS documentation](#).

### 3.4.4 SLURM

Information on installing SLURM is available in the [official SLURM documentation](#).

### 3.4.5 Flight User Suite

Information on manually installing the Flight User Suite is available in [installation documentation](#).

## 3.5 OpenFlight Action

OpenFlight Action is a platform-agnostic management tool built with a server/client architecture. The server provides a simple, yet flexible, method for defining nodes and commands. The client provides a consistent, straightforward interface for performing actions on nodes of various platforms.

Utilisation of a server/client model allows for centralised configuration and distributed control of the nodes in a cluster, no matter what platform said nodes are running on. This can allow, for example, users to be able to perform superuser-level commands on many nodes from the client without needing sudo, giving the admin a greater level of control without needing to configure sudo on multiple systems.

### 3.5.1 Server

#### Overview

**Project URL:** <https://github.com/openflighthpc/flight-action-api>

The server provides a secure, centralised location for configuring platform-specific power commands and tracking the nodes in the cluster.

#### Configuration

##### Nodes

The nodes file is used to define the nodes in the system. An example of the content for this file is in `/opt/flight/opt/action-api/config/nodes.example.yaml`, this is a useful reference for understanding the different ways that nodes can be configured.

Some example node entries are below:

```
node01:
  ranks: [metal]
  ip: 192.168.1.1

cnode01:
  ranks: [aws]
  ec2_id: i-1234567890
  aws_region: eu-west-2
```

Besides `ranks`, the key/value pairs for node entries are arbitrary and can be customised for whatever platform, use or metadata that fits your use case. The `ranks` key is used during command lookup to run rank-specific variants of the command (if available).

---

**Note:** If no ranks are present then the default version of a command will be run

---

#### Commands

Commands are stored within `/opt/flight/opt/action-api/libexec/` and are shell scripts. A command exists in a subdirectory of the aforementioned path. For example, a command called `usage` would be a directory at `/opt/flight/opt/action-api/libexec/usage` and would contain, at least, the following files:

- `metadata.yaml` - Containing command descriptions and definition

- `default.sh` - The default script to run for the commands

Additionally, this directory can contain scripts for various arbitrary rank keys, such as:

- `aws.sh` - A version of the default script with amendments made to support AWS

---

**Note:** The script files need to be executable or they will not run

---

The `metadata.yaml` file contains general command information and looks like the following:

```
help:
  summary: "Get system usage info"
  description: >
    This command gets basic system usage information and reports it back
    to the user
```

The `default.sh` file runs a simple command and reports back on the load average of the system:

```
#!/bin/bash
LOAD="$(ssh $name 'uptime')"
echo "$name load and uptime: $LOAD"
```

The `aws.sh` file includes the AWS instance ID in the script:

```
#!/bin/bash
LOAD="$(ssh $name 'uptime')"
echo "$name ($sec2_id) load and uptime: $LOAD"
```

---

**Note:** All node metadata is passed through as bash variables of the same name and case

---

## Authentication

The server utilises JWT tokens to secure client access against unauthenticated clients attempting to interact with nodes. To generate a token that's valid for 30 days, simple run:

```
flight action-api generate-token
```

This will print some information and then generate a token which can be set in the configuration of authorised clients.

## Service

OpenFlight provides a service handler that hooks non-intrusively into the system beside `systemd`. In order for action requests from clients to be properly handled by the server, the action server needs to be started:

```
flight service start action-api
```

To ensure that the action server is running after reboot, enable it:

```
flight service enable action-api
```

---

**Note:** In order for the action server to be queryable, a webserver of some kind is needed. This could be a manual Apache/Nginx setup or by using the OpenFlight WWW service (`flight service start www`). It's also worth

noting that only HTTPS is supported by the OpenFlight WWW service so ensure that it is suitably certified. The OpenFlight WWW service can assist with certificate generation, see `flight www cert-gen` for more information.

## Helpers

While the action server provides a generic framework for securely executing commands on nodes it's a fairly blank slate to begin with. To address some of the common usages of Flight Action, there are various helper packages that can be installed to provide some commands that work out-of-the-box on various cloud & metal platforms.

## Power

The OpenFlight package `flight-action-api-power` provides power management commands for multiple platforms (IPMI, AWS & Azure). The specific commands it provides are:

- `power-off` - Power off a node
- `power-on` - Power on a node
- `power-cycle` - Cycle the power, reboot the node
- `power-status` - Print the power status of the node

## Estate

The OpenFlight package `flight-action-api-estate` provides estate management commands for multiple platforms (AWS & Azure) for setting the instance size of cloud nodes. The specific commands it provides are:

- `estate-change` - Change the machine type of a node
- `estate-show` - Show the machine type of a node

## 3.5.2 Client

### Overview

**Project URL:** <https://github.com/openflighthpc/flight-action>

The Action client provides an integrated tool for communicating effectively with the API server.

### Configuration

Before the client can be used it needs to be configured to look for the right server with the correct authentication token. An example configuration file can be found at `/opt/flight/opt/action/etc/config.yaml`. reference. A simple configuration stored at `/opt/flight/opt/action/etc/config.yaml` would be something like:

```
base_url: https://gateway1/action
jwt_token: 1a2b3c4d5e6f7g8h9i0j
```

Where `base_url` is the hostname or IP address of the OpenFlight Action Server and `jwt_token` is a valid token generated on the server. If using a self-signed SSL certificate the client will fail to run unless `verify_ssl: false` is added to the configuration file.

### Command Line

The command line provides a generalised client for accessing whatever commands have been created on the server, therefore there are only a couple of consistent subcommands for the client:

- `help` - The help page will show all available commands defined in the action server
- `estate-list` - This command lists all the nodes defined in the action server

When running an action from the command line - a nodename will be needed to direct the server to run the command on the correct system. To run for multiple nodes at once, use the `-g` argument with a comma-separated list of nodes.

### Helpers

To improve accessibility and ease-of-use for the client command line, there are helpers that provide shorter endpoints for the additional content provided by the server helpers:

- `flight-power` - The endpoint `power` for managing node power state (`flight power off node01`)
- `flight-estate` - The endpoint `estate` for managing node types (`flight estate change node01`)

## 3.6 Workflow: AWS

This workflow is based on setting up the action server and client for an AWS cluster created as explained in *the Cloud AWS Workflow*.

### 3.6.1 Server Setup

#### Install Action Server

To install the action server on CentOS 7 x86\_64, simply install the package with yum:

```
yum install flight-action-api
```

---

**Note:** This assumes that the [OpenFlight Package Repositories](#) are configured on the node

---

#### Install Power Scripts

To install the power scripts to jumpstart research environment management, install with yum:

```
yum install flight-action-api-power
```

#### Install AWS CLI

For managing the power state of the nodes, the AWS CLI will need to be installed with:

```
$ yum install awscli
```

Once installed, authenticate with AWS details:

```
$ aws configure
```

## Add Nodes

Nodes are added to the config file at `/opt/flight/opt/action-api/config/nodes.yaml`, below is an example of the node configuration for a simple 3 node cluster consisting of a gateway and two compute nodes:

```
gateway1:
  ranks: [aws]
  ec2_id: i-1234567890
  aws_region: eu-west-2
node01:
  ranks: [aws]
  ec2_id: i-2345678901
  aws_region: eu-west-2
node02:
  ranks: [aws]
  ec2_id: i-3456789012
  aws_region: eu-west-2
```

## Start Service

The OpenFlight Service utility is used to start the service:

```
$ flight service start action-api
```

**Note:** The general OpenFlight webserver service also needs to be running, this can be launched with `flight service start www`

## Enable Service

The OpenFlight Service utility is used to enable the service:

```
$ flight service enable action-api
```

**Note:** In order to integrate the service enabling with systemd (to ensure the service starts at boot), the `flight-plugin-system-systemd-service` package will need to be installed

## Generate Token

To generate a 30-day authentication token for the client, run:

```
$ flight action-api generate-token
eyJhbGciOiJIUzI1NiJ9.eyJhZG1pbWVsbCwiZm9udGVzODcyNjA2fQ.WXm-
↪F07bA178UTZHAsFKLXzYokBwVfgMabzIMgNfc5Y
```

This will return a token that can be entered into the config of a power client to authenticate against this server.

## 3.6.2 Client Setup

### Install Action Client

To install the action client on CentOS 7 x86\_64, simply install the package with yum:

```
yum install flight-action
```

---

**Note:** This assumes that the [OpenFlight Package Repositories](#) are configured on the node

---

### Add Security Token

Add security token to `/opt/flight/opt/action/etc/config.yaml` so the file looks similar to the following:

```
base_url: https://gateway1/action
jwt_token: eyJhbGciOiJIUzI1NiJ9.eyJhZG1pbI6bnVsbCwiZXhwIjoxNTgxODcyNjA2fQ.WXm-
↪F07bAl78UTZHASFKLXzYokBwVfgMabzIMgNfc5Y
```

---

**Note:** If using a self-signed certificate then add `verify_ssl: false` to the config

---

### List Available Nodes

A list of the configured nodes can be seen from the client as so:

```
$ flight action estate-list
gateway1
node01
node02
```

---

**Note:** The `flight` command can be executed with its full path (`/opt/flight/bin/flight`) in the case that `flight-starter` has not been installed to add the command to the environment.

---

### Check Power Status of Nodes

To check the power status of all nodes, simply:

```
$ flight power status -g gateway1,node0[1-2]
gateway1: ON
node01: ON
node02: ON
```

### Power Off a Node

A node can be powered off as follows:

```
$ flight power off node02
Power off node02, confirm to proceed [y/n]?
y
OK
```

### Power On a Node

A node can be powered on as follows:

```
$ flight power on node02
OK
```

### Restart a Node

To power cycle a node, simply:

```
$ flight power cycle node02
Power cycle node02, confirm to proceed [y/n]?
y
OK
```

## 3.7 Workflow: Azure

This workflow is based on setting up the power server and client for an Azure cluster created as explained in *the Cloud Azure Workflow*

### 3.7.1 Server Setup

#### Install Action Server

To install the action server on CentOS 7 x86\_64, simply install the package with yum:

```
yum install flight-action-api
```

---

**Note:** This assumes that the [OpenFlight Package Repositories](#) are configured on the node

---

#### Install Power Scripts

To install the power scripts to jumpstart research environment management, install with yum:

```
yum install flight-action-api-power
```

### Install Azure CLI

Refer to the [Azure CLI documentation](#) for help with installing the repository and tool.

Once installed, authenticate the tool with your Azure account:

```
$ az login
```

### Add Nodes

Nodes are added to the config file at `/opt/flight/opt/action-api/config/nodes.yaml`, below is an example of the node configuration for a simple 3 node cluster consisting of a gateway and two compute nodes:

```
gateway1:
  ranks: [azure]
  azure_resource_group: mycluster-abcd123
  azure_name: flightcloudclustergateway1
node01:
  ranks: [azure]
  azure_resource_group: mycluster-abcd123
  azure_name: flightcloudclusternode01
node02:
  ranks: [azure]
  azure_resource_group: mycluster-abcd123
  azure_name: flightcloudclusternode02
```

### Start Service

The OpenFlight Service utility is used to start the service:

```
$ flight service start action-api
```

---

**Note:** The general OpenFlight webserver service also needs to be running, this can be launched with `flight service start www`

---

### Enable Service

The OpenFlight Service utility is used to enable the service:

```
$ flight service enable action-api
```

---

**Note:** In order to integrate the service enabling with systemd (to ensure the service starts at boot), the `flight-plugin-system-systemd-service` package will need to be installed

---

### Generate Token

To generate a 30-day authentication token for the client, run:

```
$ flight action-api generate-token
eyJhbGciOiJIUzI1NiJ9.eyJhZG1pbI6bnVsbCwiZXhwIjoxNTgxODcyNjA2fQ.WXm-
↪F07bAl78UTZHAsFKLXzYokBwVfgMabzIMgNfc5Y
```

This will return a token that can be entered into the config of a power client to authenticate against this server.

## 3.7.2 Client Setup

### Install Action Client

To install the action client on CentOS 7 x86\_64, simply install the package with yum:

```
yum install flight-action
```

**Note:** This assumes that the [OpenFlight Package Repositories](#) are configured on the node

### Add Security Token

Add security token to `/opt/flight/opt/action/etc/config.yaml` so the file looks similar to the following:

```
base_url: https://gateway1/action
jwt_token: eyJhbGciOiJIUzI1NiJ9.eyJhZG1pbI6bnVsbCwiZXhwIjoxNTgxODcyNjA2fQ.WXm-
↪F07bAl78UTZHAsFKLXzYokBwVfgMabzIMgNfc5Y
```

**Note:** If using a self-signed certificate then add `verify_ssl: false` to the config

### List Available Nodes

A list of the configured nodes can be seen from the client as so:

```
$ flight action estate-list
gateway1
node01
node02
```

**Note:** The `flight` command can be executed with its full path (`/opt/flight/bin/flight`) in the case that `flight-starter` has not been installed to add the command to the environment.

### Check Power Status of Nodes

To check the power status of all nodes, simply:

```
$ flight power status -g gateway1,node0[1-2]
gateway1: ON
node01: ON
node02: ON
```

### Power Off a Node

A node can be powered off as follows:

```
$ flight power off node02
Power off node02, confirm to proceed [y/n]?
y
OK
```

### Power On a Node

A node can be powered on as follows:

```
$ flight power on node02
OK
```

### Restart a Node

To power cycle a node, simply:

```
$ flight power cycle node02
Power cycle node02, confirm to proceed [y/n]?
y
OK
```

## 3.8 Basic Research Environment Operation

### 3.8.1 Logging in

You can access the login node for your private Flight Compute research environment using SSH to connect to the externally facing IP address of the login node. You will need to use the SSH keypair configured for the research environment in order to access it.

When you login to the research environment via SSH, you are automatically placed in your home-directory. This area is shared across all compute nodes in the research environment, and is mounted in the same place on every compute. Data copied to the research environment or created in your home-directory on the login node is also accessible from all compute nodes.

#### Linux/Mac

To access the research environment login node from a Linux or Mac client, use the following command:

- `ssh -i mypublickey.pem flight@52.50.141.144`

**Where:**

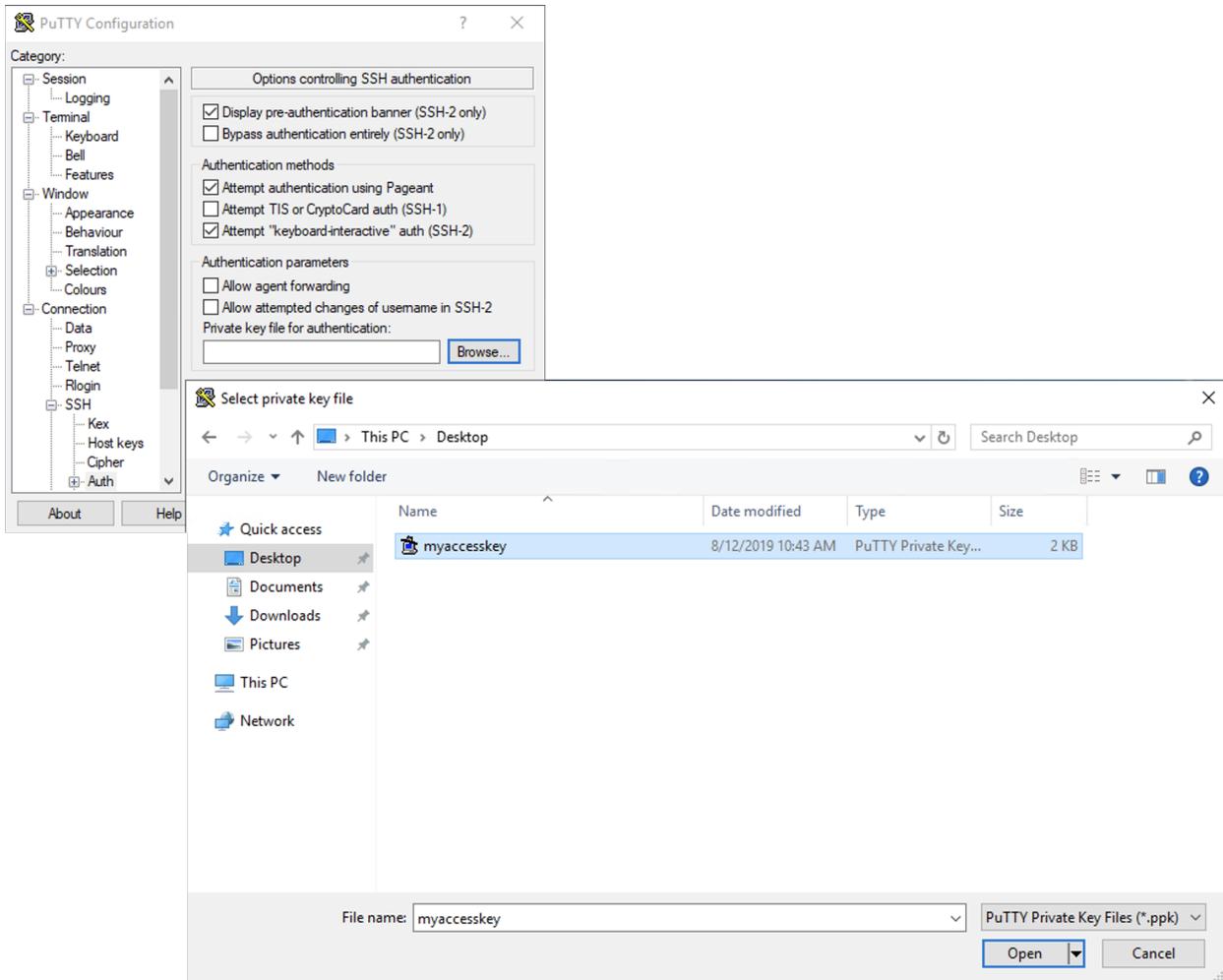
- `mypublickey.pem` is the name of your private key associated with the public SSH key you set when launching the research environment
- `flight` is the username of the user on the research environment
- `52.50.141.144` is the Access-IP address for the gateway node of the research environment

## Windows

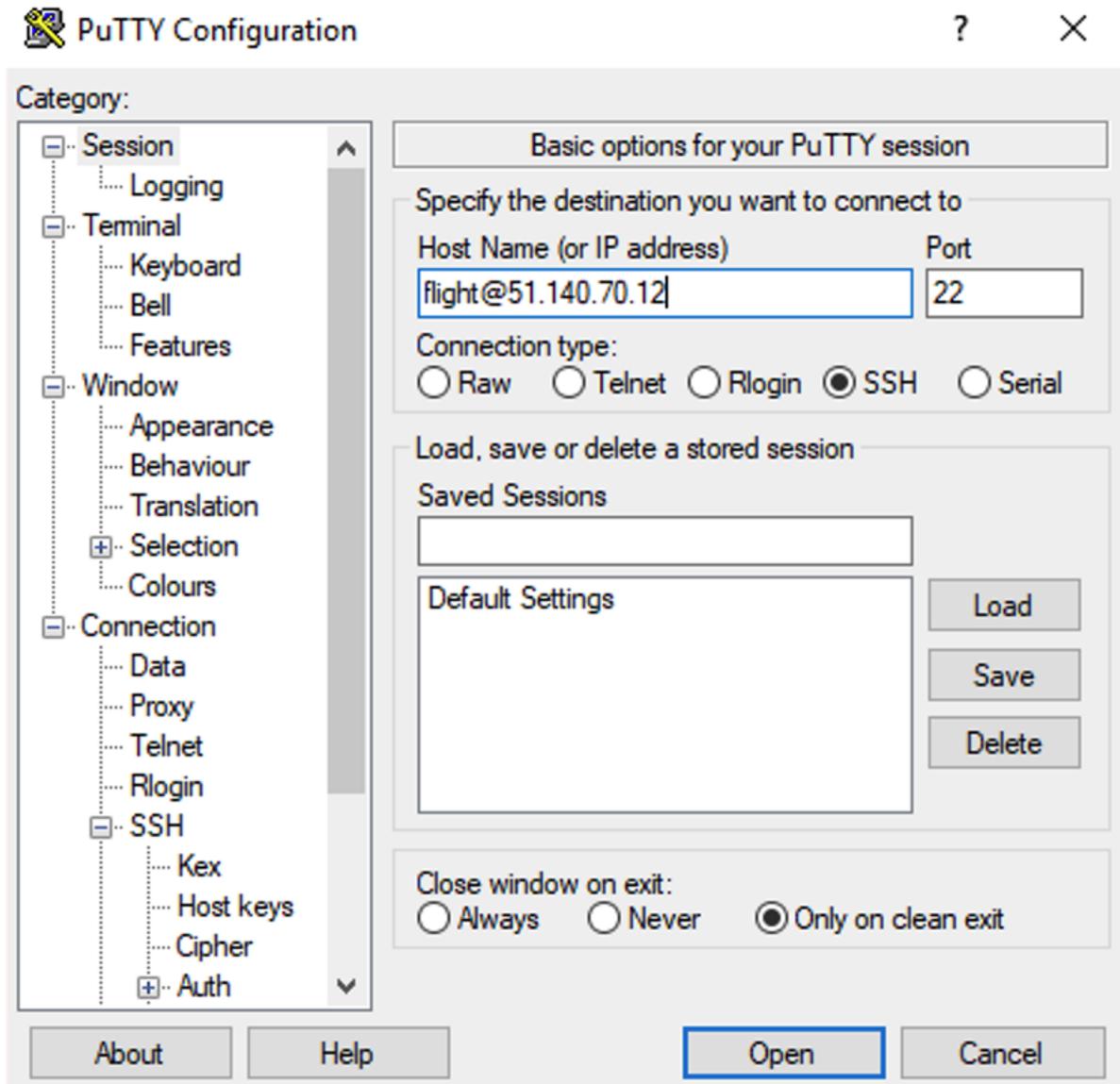
If you are accessing from a Windows client using the Putty utility, the private key associated with the account will need to be converted to ppk format from pem to be compatible with Putty. This can be done as follows:

- Open PuTTYgen (this will already be installed on your system if Putty was installed using .msi and not launched from the .exe - if you do not think you have this, download putty-installer from here <http://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>)
- Select *Conversions -> Import Key*
- Locate `.pem` file and click *open*
- Click *Save Private Key*
- Answer *Yes* to saving without a passphrase
- Input the name for the newly generated ppk to be saved as

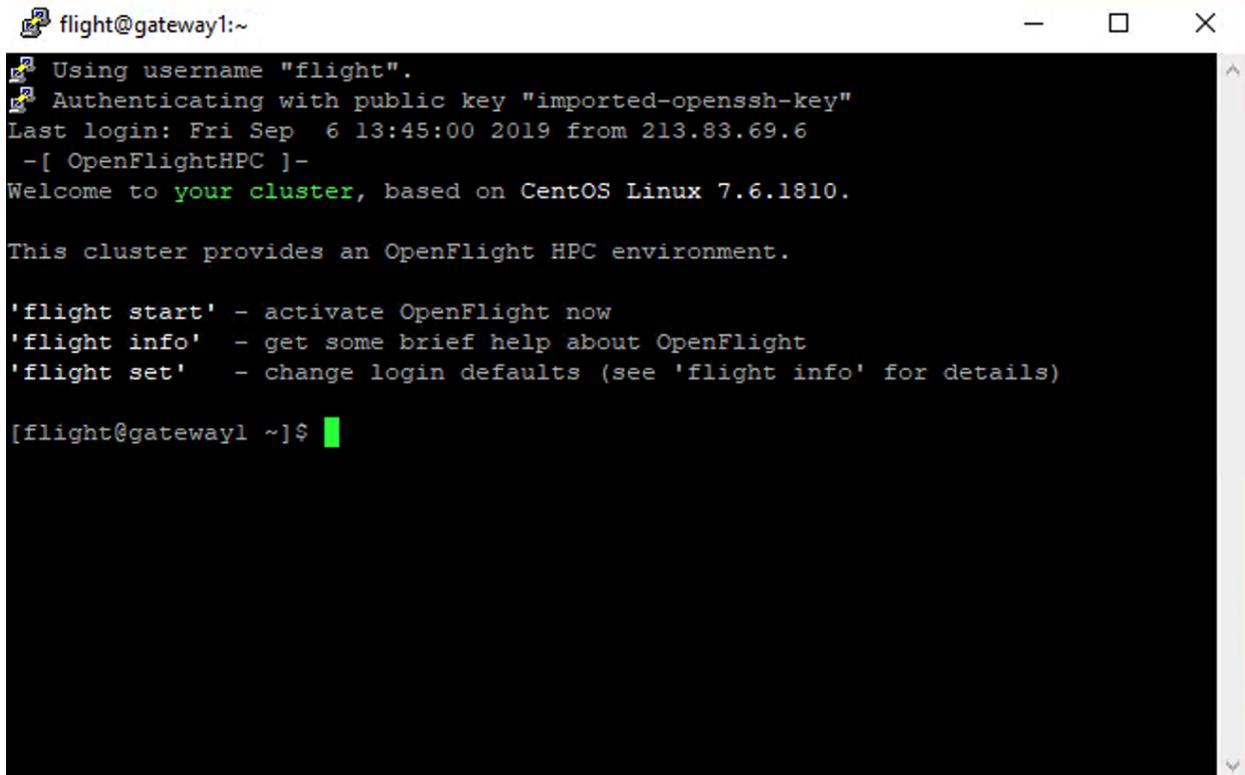
To load the key in Putty, select *Connection -> SSH -> Auth*, click *Browse* and select the ppk that was generated from the above steps.



Next, enter the username and IP address of the research environment login node in the “Host Name” box provided (in the *Session* section):



The first time you connect to your research environment, you will be prompted to accept a new server SSH hostkey. This happens because you've never logged in to your research environment before - it should only happen the first time you login; click **OK** to accept the warning. Once connected to the research environment, you should be logged in to the research environment login node as your user.

A terminal window titled 'flight@gateway1:~' with standard window controls. The terminal output shows the user logging in as 'flight' using a public key. The last login was on Fri Sep 6 13:45:00 2019 from 213.83.69.6. The prompt is '-[ OpenFlightHPC ]-' and the user is welcomed to 'your cluster' based on CentOS Linux 7.6.1810. A message states 'This cluster provides an OpenFlight HPC environment.' followed by a list of commands: 'flight start' (activate OpenFlight now), 'flight info' (get some brief help about OpenFlight), and 'flight set' (change login defaults). The prompt is now '[flight@gateway1 ~]\$' with a green cursor.

```
flight@gateway1:~  
Using username "flight".  
Authenticating with public key "imported-openssh-key"  
Last login: Fri Sep  6 13:45:00 2019 from 213.83.69.6  
-[ OpenFlightHPC ]-  
Welcome to your cluster, based on CentOS Linux 7.6.1810.  
  
This cluster provides an OpenFlight HPC environment.  
  
'flight start' - activate OpenFlight now  
'flight info'  - get some brief help about OpenFlight  
'flight set'   - change login defaults (see 'flight info' for details)  
  
[flight@gateway1 ~]$
```

### 3.8.2 Becoming the root user

Most research environment operations, including starting applications and running jobs, should be performed as the user created when the Flight Compute research environment was launched from the launch template. However - for some privileged operations, users may need to change to being the root user. Users can prefix any command they want to run as the root user with the `sudo` command; e.g.

```
sudo yum install screen
```

For security reasons, SSH login as the root user is not permitted to a Flight Compute environment. To get a Linux shell with root privileges, please login as your standard user then execute the command `sudo -s`.

**Warning:** Users must exercise caution when running commands as root, as they have the potential to disrupt research environment operations.

### 3.8.3 Moving between login and compute nodes

OpenFlight Compute research environments automatically configure a trust relationship between login and compute nodes in the same research environment to allow users to login between nodes via SSH without a password. This configuration allows moving quickly and easily between nodes, and simplifies running large-scale jobs that involve multiple nodes. From the command line, a user can simply use the `ssh <node-name>` command to login to one of the compute nodes from the login node. For example, to login to a compute node named `node01` from the login node, use the command:

```
ssh node01
```

Use the `logout` command (or press **CTRL+D**) to exit the compute node and return to the login node.

## 3.9 Working with Data and Files

### 3.9.1 Organising Data on Your Research Environment

#### Shared filesystem

Your OpenFlight Compute research environment includes a shared home filesystem which is mounted across the login and all compute nodes. Files copied to this area are available via the same absolute path on all research environment nodes. The shared filesystem is typically used for job-scripts, input and output data for the jobs you run.

---

**Note:** If running a single node research environment then the home directory is not shared over NFS as there are no other resources to share data with

---

#### Your home directory

The shared filesystem includes the home-directory area for the `flight` user which is created when your research environment is launched. Linux automatically places users in their home-directory when they login to a node. By default, Flight Compute will create your home-directory under the `/users/` directory, named `flight` (`/users/flight`).

The Linux command line will accept the `~` (*tilde*) symbol as a substitute for the currently logged-in users' home-directory. The environment variable `$HOME` is also set to this value by default. Hence, the following three commands are all equivalent when logged in as the user **flight**:

- `ls /users/flight`
- `ls ~`
- `ls $HOME`

The **root** user in Linux has special meaning as a privileged user, and does not have a shared home-directory across the research environment. The **root** account on all nodes has a home-directory in `/root`, which is separate for every node. For security reasons, users are not permitted to login to a node as the root user directly - please login as a standard user and use the `sudo` command to get privileged access.

#### Local scratch storage

Your compute nodes have an amount of disk space available to store temporary data under the `/tmp` mount-point. This area is intended for temporary data created during compute jobs, and shouldn't be used for long-term data storage. Compute nodes are configured to clear up temporary space automatically, removing orphan data left behind by jobs.

Users must make sure that they copy data they want to keep back to the shared filesystem after compute jobs have been completed.

#### Copying data between nodes

---

**Note:** This isn't applicable to a single node research environment

---

Flight Compute research environment login and compute nodes all mount the shared filesystem, so it is not normally necessary to copy data directly between nodes in the research environment. Users simply need to place the data to be shared in their home-directory on the login node, and it will be available on all compute nodes in the same location.

If necessary, users can use the `scp` command to copy files from the compute nodes to the login node; for example:

- `scp node01:/tmp/myfile.txt .`

Alternatively, users could login to the compute node (e.g. `ssh node01`) and copy the data back to the shared filesystem on the node:

```
ssh node01
cp /tmp/myfile ~/myfile
```

### 3.9.2 Copying data files to the research environment

Many compute workloads involve processing data on the research environment - users often need to copy data files to the research environment for processing, and retrieve processed data and results afterwards. This documentation describes a number of methods of working with data on your research environment, depending on how users prefer to transfer it.

#### Using command-line tools to copy data

The research environment login node is accessible via SSH, allowing use of the `scp` and `sftp` commands to transfer data from your local client machine.

##### Linux/Mac

Linux and Mac users can use in-built SSH support to copy files. To copy file **mydata.zip** to your research environment on IP address 52.48.62.34, use the command:

```
scp -i mykeyfile.pem mydata.zip flight@52.48.62.34:.
```

- replace `mykeyfile.pem` with the name of your SSH public key
- replace `flight` with your username on the research environment

##### Windows

Windows users can download and install the `pscp` command to perform the same operation (for this you will need your `.pem` key in `.ppk` format, see *connecting from Windows with Putty*):

```
pscp -i mykeyfile.ppk mydata.zip flight@52.48.62.34:/users/flight/.
```

##### SCP/PSCP

Both the `scp` and the `pscp` commands take the parameter `-r` to recursively copy entire directories of files to the research environment.

To retrieve files from the research environment, simply specify the location of the remote file first in the `scp` command, followed by the location on the local system to put the file; e.g.

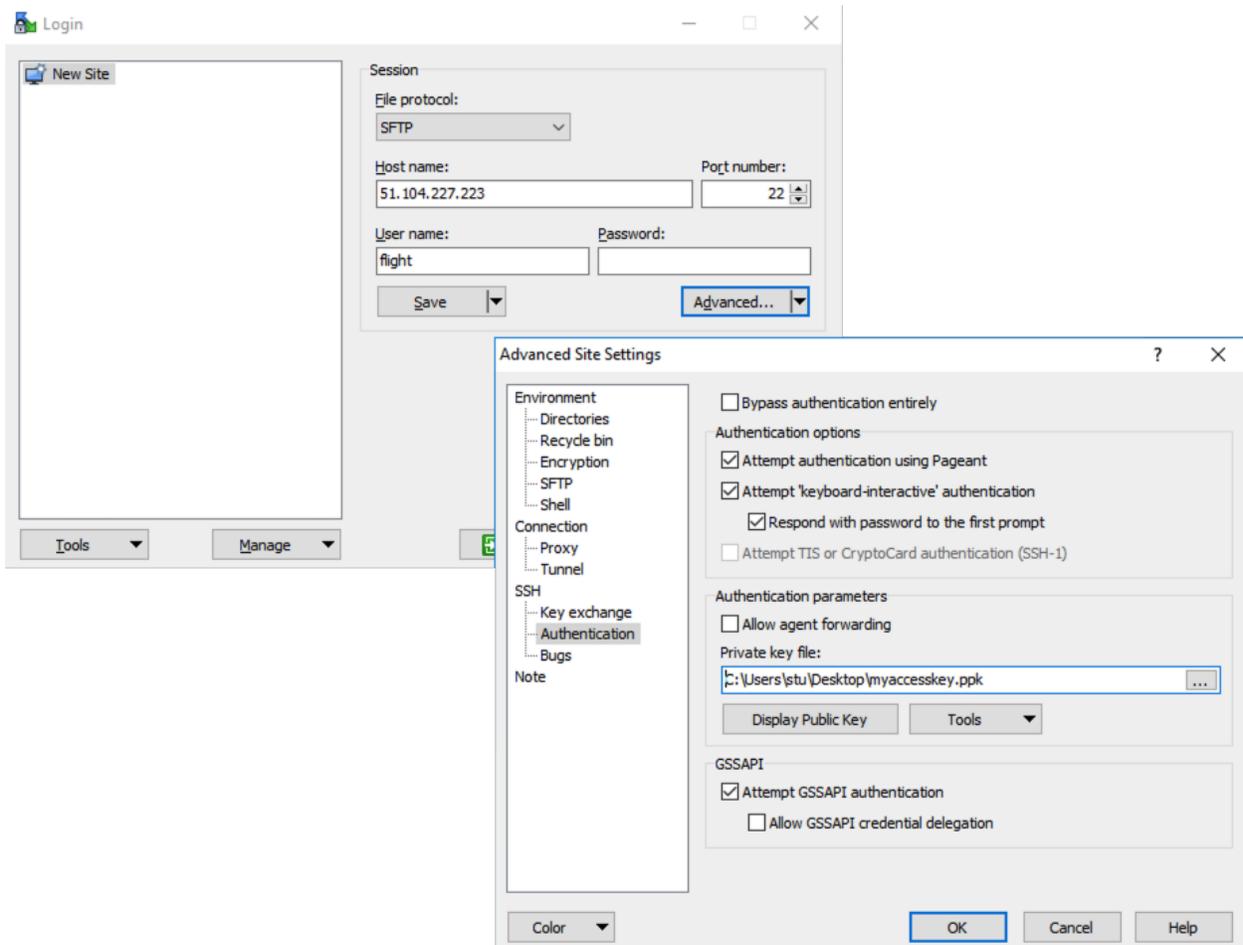
To copy file **myresults.zip** from your research environment on IP address 52.48.62.34 to your local Linux or Mac client:

```
scp -i mykeyfile.pem flight@52.48.62.34:/users/flight/myresults.zip .
```

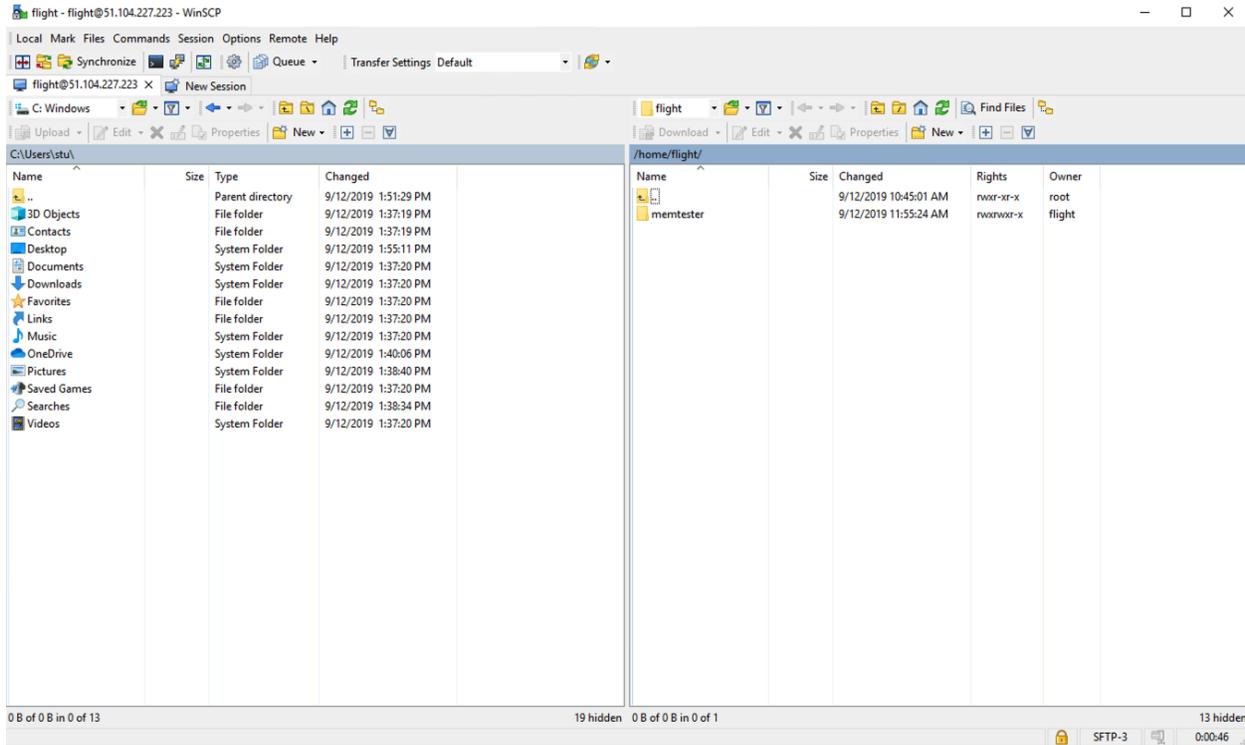
#### Using a graphical client to copy data

There are also a number of graphical file-management interfaces available that support the SSH/SCP/SFTP protocols. A graphical interface can make it easier for new users to manage their data, as they provide a simple drag-and-drop interface that helps to visualise where data is being stored. The example below shows how to configure the **WinSCP** utility on a Windows client to allow data to be moved to and from a research environment.

- On a Windows client, download and install [WinSCP](#)
- Start WinSCP; in the **login** configuration box, enter the IP address of your Flight Compute research environment login node in the `Host name` box
- Enter the username you configured for your research environment in the `User name` box (the default user is `flight`)
- Click on the `Advanced` box and navigate to the `SSH` sub-menu, and the `Authentication` item
- In the `Private key file` box, select your research environment access private key, and click the `OK` box.



- Optionally click the `Save` button and give this session a name
- Click the `Login` button to connect to your research environment
- Accept the warning about adding a new server key to your cache; this message is displayed only once when you first connect to a new research environment
- WinSCP will login to your research environment; the window shows your local client machine on the left, and the research environment on the right
- To copy files to the research environment from your client, click and drag them from the left-hand window and drop them on the right-hand window
- To copy files from the research environment to your client, click and drag them from the right-hand window and drop them on the left-hand window



The amount of time taken to copy data to and from your research environment will depend on a number of factors, including:

- The size of the data being copied
- The speed of your Internet link to the research environment; if you are copying large amounts of data, try to connect using a wired connection rather than wireless
- The type and location of your research environment login node instance

### Object storage for archiving data

As an alternative to copying data back to your client machine, users may prefer to upload their data to a cloud-based object storage service instead. Cloud storage solutions such as [AWS S3](#), [Dropbox](#) and [SWIFT](#) have command-line tools which can be used to connect existing cloud storage to your research environment. Benefits of using an object-based storage service include:

- Data is kept safe and does not have to be independently backed-up
- Storage is easily scalable, with the ability for data to grow to practically any size
- You only pay for what you use; you do not need to buy expansion room in advance
- Storage service providers often have multiple tiers available, helping to reduce the cost of storing data
- Data storage and retrieval times may be improved, as storage service providers typically have more bandwidth than individual sites
- Your company, institution or facility may receive some storage capacity for free which you could use

Object storage is particularly useful for archiving data, as it typically provides a convenient, accessible method of storing data which may need to be shared with a wide group of individuals.

## Saving data before terminating your research environment

When you've finished working with your OpenFlight Flight Compute research environment, you can select to terminate it in the console for your Cloud service. This will stop any running instances and wipe the shared storage area before returning the block storage volumes back to the provider. Before you shutdown your research environment, users must ensure that they store their data safely in a persistent service, using one of the methods described in this documentation. When you next launch a Flight Compute research environment, you can restore your data from the storage service to begin processing again.

## 3.10 Genders and PDSH

---

**Note:** Genders & PDSH functionality is not available or useful on a single node research environment, this page only applies to multi-node research environments

---

A combination of genders and pdsh can allow for management and monitoring of multiple nodes at a time. OpenFlight provides a build of PDSH that integrates with the rest of the User Suite.

### 3.10.1 Installing nodeattr and pdsh

Nodeattr and pdsh can be installed using the yum package manager, simply:

```
sudo yum -y install flight-pdsh
```

Once installed, the pdsh command will be available to active OpenFlight User Suite sessions (use `flight start` to activate the session). The priority of the OpenFlight pdsh command in the path can be configured with:

```
flight config set pdsh.priority X
```

Where X is one of:

- `system` - Prefer system PDSH
- `embedded` - Prefer OpenFlight PDSH
- `disabled` - Do not include OpenFlight PDSH in PATH

---

**Note:** OpenFlight PDSH defaults `system` priority, to ensure that the OpenFlight User Environment uses the OpenFlight PDSH, set the priority to `embedded` after installation (`flight config set pdsh.priority embedded`)

---

### 3.10.2 Finding the names of your compute nodes

An OpenFlight Compute research environment may contain any number of compute nodes depending on your research environment size. The hostnames of compute nodes usually follow a sequential order (e.g. `node01`, `node02`, `node03`... `node10`). OpenFlight Compute automatically creates a list of compute node names and uses them to populate a *genders* group called **nodes**. This genders file can be found at `/opt/flight/etc/genders`.

Users can find the names of their compute nodes by using the `nodeattr` command; e.g.

- `nodeattr -s nodes`

- shows a space-separated list of current compute node hostnames
- **nodeattr -c nodes**
  - shows a comma-separated list of current compute node hostnames
- **nodeattr -n nodes**
  - shows a new-line-separated list of current compute node hostnames

The login node hostname for Flight Compute research environments launched using default templates is always `gateway1`.

### 3.10.3 Using PDSH

Users can run a command across all compute nodes at once using the `pdsh` command. This can be useful if users want to make a change to all nodes in the research environment - for example, installing a new software package. The `pdsh` command can take a number of parameters that control how commands are processed; for example:

- **pdsh -g all uptime**
  - executes the `uptime` command on all available compute and login nodes in the research environment
- **pdsh -g nodes 'sudo yum -y install screen'**
  - use `yum` to install the `screen` package as the root user on all compute nodes
- **pdsh -g nodes -f 1 df -h /tmp**
  - executes the command `df -h /tmp` on all compute nodes of the research environment, one at a time (`fanout=1`)
- **pdsh -w node01,node03 which ldconfig**
  - runs the `which ldconfig` command on two named nodes only

## 3.11 Local Jobs

### 3.11.1 What is a Job?

A job can be loosely defined as an automated research task, for example, a bash script that runs various stages in an OpenFoam simulation on a model.

Jobs vary in size, resource usage and run time. A job could utilise multiple cores through parallel libraries or simply run on a single core.

### 3.11.2 Why Run a Local Job?

When using a personal research environment there isn't a need to monitor resource usage as closely as multi-user systems where miscommunication and overloaded resources can negatively impact research progress. With all the resources available to one user it is quicker and easier to run jobs simply through a terminal than using a queue system.

Local jobs also have the benefit over schedulers by launching immediately and providing all output through a single terminal.

### 3.11.3 Running a Local Job

Local job scripts can be written in any language supported within the research environment. In most cases this is likely to be bash as it's a flexible and functional shell scripting language which enables users to intuitively navigate the filesystem, launch applications and manage data output.

In the event that a job script is executable, contains a shebang specifying the launch language and is in the current directory, it's as simple to run as:

```
[flight@gateway1 ~]$ ./myjob.sh
```

## 3.12 What is a Scheduler?

### 3.12.1 What is a batch job scheduler?

Most existing High-performance Compute Research Environments are managed by a job scheduler; also known as the batch scheduler, workload manager, queuing system or load-balancer. The scheduler allows multiple users to fairly share compute nodes, allowing system administrators to control how resources are made available to different groups of users. All schedulers are designed to perform the following functions:

- Allow users to submit new jobs to the research environment
- Allow users to monitor the state of their queued and running jobs
- Allow users and system administrators to control running jobs
- Monitor the status of managed resources including system load, memory available, etc.

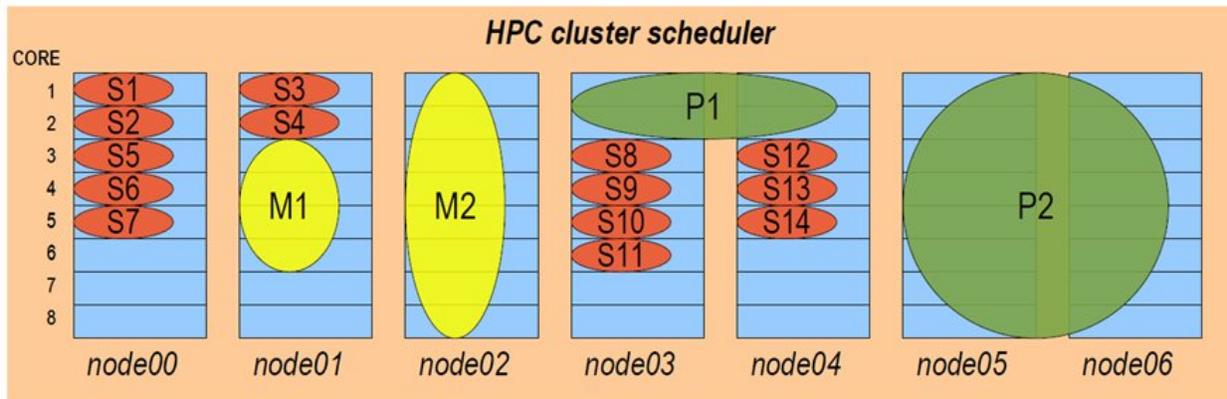
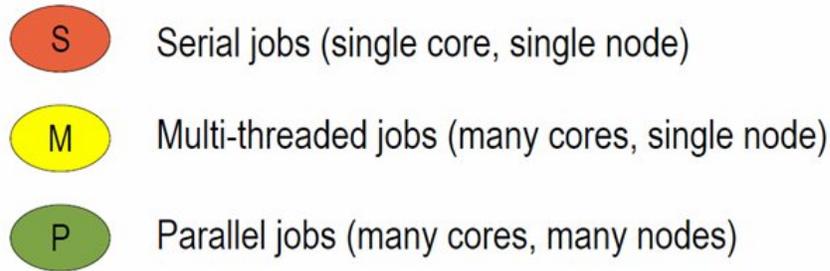
When a new job is submitted by a user, the research environment scheduler software assigns compute cores and memory to satisfy the job requirements. If suitable resources are not available to run the job, the scheduler adds the job to a queue until enough resources are available for the job to run. You can configure the scheduler to control how jobs are selected from the queue and executed on research environment nodes, including automatically preparing nodes to run parallel MPI jobs. Once a job has finished running, the scheduler returns the resources used by the job to the pool of free resources, ready to run another user job.

### 3.12.2 Types of compute job

Users can run a number of different types of job via the research environment scheduler, including:

- **Batch jobs**; single-threaded applications that run only on one compute core
- **Array jobs**; two or more similar batch jobs which are submitted together for convenience
- **SMP** or **multi-threaded** jobs; multi-threaded applications that run on two or more compute cores on the same compute node
- **Parallel jobs**; multi-threaded applications making use of an MPI library to run on multiple cores spread over one or more compute nodes

The research environment job-scheduler is responsible for finding compute nodes in your research environment to run all these different types of jobs on. It keeps track of the available resources and allocates jobs to individual groups of nodes, making sure not to over-commit CPU and memory. The example below shows how a job-scheduler might allocate jobs of different types to a group of 8-CPU-core compute nodes:



### 3.12.3 Interactive and batch jobs

Users typically interact with compute research environments by running either **interactive** or **batch** (also known as **non-interactive**) jobs.

- An interactive job is one that the user directly controls, either via a graphical interface or by typing at the command-prompt.
- A batch job is run by writing a list of instructions that are passed to compute nodes to run at some point in the future.

Both methods of running jobs can be equally as efficient, particularly on a personal, ephemeral research environment. Both classes of job can be of any type - for example, it's possible to run interactive parallel jobs and batch multi-threaded jobs across your research environment. The choice of which class of job-type you want to use will depend on the application you're running, and which method is more convenient for you to use.

### 3.12.4 Why use a job-scheduler on a personal research environment?

Good question. On shared multi-user research environments, a job-scheduler is often used as a control mechanism to make sure that users don't unfairly monopolise the valuable compute resources. In extreme cases, the scheduler may be wielded by system administrators to force "good behaviour" in a shared environment, and can feel like an imposition to research environment users.

With your own personal research environment, you have the ability to directly control the resources available for your job - you don't need a job-scheduler to limit your usage.

However - there are a number of reasons why your own job-scheduler can still be a useful tool in your research environment:

1. It can help you organise multi-stage work flows, with batch jobs launching subsequent jobs in a defined process.
2. It can automate launching of MPI jobs, finding available nodes to run applications on.

3. It can help prevent accidentally over-allocating CPUs or memory, which could lead to nodes failing.
4. It can help bring discipline to the environment, providing a consistent method to replicate the running of jobs in different environments.
5. Jobs queued in the scheduler can be used to trigger scaling-up the size of your research environment, with compute nodes released from the research environment when there are no jobs to run, saving you money.

Your OpenFlight Flight Compute research environment comes with a job-scheduler pre-installed, ready for you to start using. The scheduler uses very few resources when idle, so you can choose to use it if you find it useful, or run jobs manually across your research environment if you prefer.

## 3.13 Slurm Scheduler

The **Slurm** research environment job-scheduler is an open-source project used by many high performance computing systems around the world - including many of the [TOP 500](#) supercomputers.

### 3.13.1 Running an interactive job

---

**Note:** If using a single node research environment with a scheduler then interactive jobs are unnecessary as the only available resources will be local

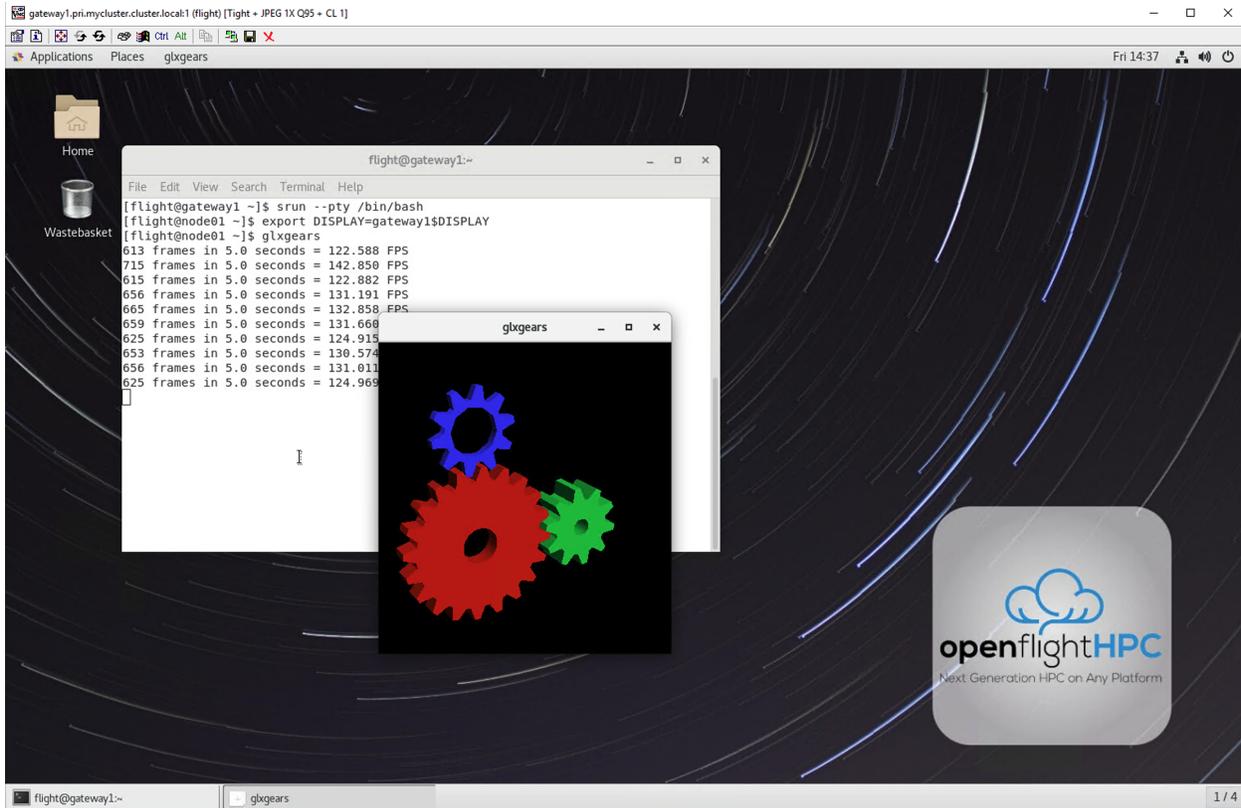
---

You can start a new interactive job on your Flight Compute research environment by using the `srun` command; the scheduler will search for an available compute node, and provide you with an interactive login shell on the node if one is available.

```
[centos@gateway1 (scooby) ~]$ srun --pty /bin/bash
[centos@node01 (scooby) ~]$
[centos@node01 (scooby) ~]$ squeue
      JOBID PARTITION    NAME   USER  ST       TIME  NODES NODELIST(REASON)
         3         all     bash   centos R        0:39      1 node01
```

In the above example, the `srun` command is used together with two options: `--pty` and `/bin/bash`. The `--pty` option executes the task in pseudo terminal mode, allowing the session to act like a standard terminal session. The `/bin/bash` option is the command that you wish to run - here the default Linux shell, `BASH`.

Alternatively, the `srun` command can also be executed from an interactive desktop session; the job-scheduler will automatically find an available compute node to launch the job on. Applications launched from within the `srun` session are executed on the assigned research environment compute node.



**Note:** The Slurm scheduler does not automatically set up your session to allow you to run graphical applications inside an interactive session. Once your interactive session has started, you must run the following command before running a graphical application: `export DISPLAY=gateway1$DISPLAY`

**Warning:** Running X applications from a compute node may not work due to missing X libraries on the compute node, these can be installed from an SSH session into a compute node with `sudo yum groupinstall "X Window System"`

When you've finished running your application in your interactive session, simply type `logout`, or press **Ctrl+D** to exit the interactive job.

If the job-scheduler could not satisfy the resource you've requested for your interactive job (e.g. all your available compute nodes are busy running other jobs), it will report back after a few seconds with an error:

```
[centos@gateway1 (scooby) ~]$ srun --pty /bin/bash
srun: job 20 queued and waiting for resources
```

### 3.13.2 Submitting a batch job

Batch (or non-interactive) jobs allow users to leverage one of the main benefits of having a research environment scheduler; jobs can be queued up with instructions on how to run them and then executed across the research environment while the user *does something else*. Users submit jobs as scripts, which include instructions on how to run the job - the output of the job (*stdout* and *stderr* in Linux terminology) is written to a file on disk for review later on. You can write a batch job that does anything that can be typed on the command-line.

We'll start with a basic example - the following script is written in bash (the default Linux command-line interpreter). You can create the script yourself using the [Nano](#) command-line editor - use the command `nano simplejobscript.sh` to create a new file, then type in the contents below. The script does nothing more than print some messages to the screen (the **echo** lines), and sleeps for 120 seconds. We've saved the script to a file called `simplejobscript.sh` - the `.sh` extension helps to remind us that this is a *shell* script, but adding a filename extension isn't strictly necessary for Linux.

```
#!/bin/bash -l
echo "Starting running on host $HOSTNAME"
sleep 120
echo "Finished running - goodbye from $HOSTNAME"
```

**Note:** We use the `-l` option to bash on the first line of the script to request a login session. This ensures that environment modules can be loaded as required as part of your script.

We can execute that script directly on the login node by using the command `bash simplejobscript.sh` - after a couple of minutes, we get the following output:

```
Started running on host gateway1
Finished running - goodbye from gateway1
```

To submit your job script to the research environment job scheduler, use the command `sbatch simplejobscript.sh`. The job scheduler should immediately report the job-ID for your job; your job-ID is unique for your current OpenFlight Flight Compute research environment - it will never be repeated once used.

```
[centos@gateway1 (scooby) ~]$ sbatch simplejobscript.sh
Submitted batch job 21

[centos@gateway1 (scooby) ~]$ ls
simplejobscript.sh  slurm-21.out

[centos@gateway1 (scooby) ~]$ cat slurm-21.out
Starting running on host node01
Finished running - goodbye from node01
```

### 3.13.3 Viewing and controlling queued jobs

Once your job has been submitted, use the `squeue` command to view the status of the job queue. If you have available compute nodes, your job should be shown in the **R** (running) state; if your compute nodes are busy, or you've launched an auto-scaling research environment and currently have no running nodes, your job may be shown in the **PD** (pending) state until compute nodes are available to run it. If a job is in **PD** state - the reason for being unable to run will be displayed in the **NODELIST (REASON)** column of the `squeue` output.

```
[centos@gateway1 (scooby) ~]$ squeue
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
41	all	simplejo	centos	R	0:03	1	node01
42	all	simplejo	centos	R	0:00	1	node01

You can keep running the `squeue` command until your job finishes running and disappears from the queue. The output of your batch job will be stored in a file for you to look at. The default location to store the output file is your home directory. You can use the Linux `more` command to view your output file:

```
[centos@gateway1 (scooby) ~]$ more slurm-42.out
Starting running on host node01
Finished running - goodbye from node01
```

Your job runs on whatever node the scheduler can find which is available for use - you can try submitting a bunch of jobs at the same time, and using the `squeue` command to see where they run. The scheduler is likely to spread them around over different nodes (if you have multiple nodes). The login node is not included in your research environment for scheduling purposes - jobs submitted to the scheduler will only be run on your research environment compute nodes. You can use the `scancel <job-ID>` command to delete a job you've submitted, whether it's running or still in the queued state.

```
[centos@gateway1 (scooby) ~]$ sbatch simplejobscrip.sh
Submitted batch job 46
[centos@gateway1 (scooby) ~]$ sbatch simplejobscrip.sh
Submitted batch job 47
[centos@gateway1 (scooby) ~]$ sbatch simplejobscrip.sh
Submitted batch job 48
[centos@gateway1 (scooby) ~]$ squeue
      JOBID PARTITION   NAME   USER  ST       TIME  NODES NODELIST(REASON)
       43          all simplejo centos  R        0:04      1 node01
       44          all simplejo centos  R        0:04      1 node01
       45          all simplejo centos  R        0:04      1 node02
       46          all simplejo centos  R        0:04      1 node02
       47          all simplejo centos  R        0:04      1 node03
       48          all simplejo centos  R        0:04      1 node03

[centos@gateway1 (scooby) ~]$ scancel 47
[centos@gateway1 (scooby) ~]$ squeue
      JOBID PARTITION   NAME   USER  ST       TIME  NODES NODELIST(REASON)
       43          all simplejo centos  R        0:11      1 node01
       44          all simplejo centos  R        0:11      1 node01
       45          all simplejo centos  R        0:11      1 node02
       46          all simplejo centos  R        0:11      1 node02
       48          all simplejo centos  R        0:11      1 node03
```

### 3.13.4 Viewing compute host status

Users can use the `sinfo -Nl` command to view the status of compute node hosts in your Flight Compute research environment.

```
[centos@gateway1 (scooby) ~]$ sinfo -Nl
Fri Aug 26 14:46:34 2016
NODELIST      NODES PARTITION      STATE CPUS      S:C:T MEMORY TMP_DISK WEIGHT_
↪AVAIL_FE REASON
node01        1      all*      idle   2      2:1:1   3602   20462      1  (null)
↪none
node02        1      all*      idle   2      2:1:1   3602   20462      1  (null) none
node03        1      all*      idle   2      2:1:1   3602   20462      1  (null) none
node04        1      all*      idle   2      2:1:1   3602   20462      1  (null) none
node05        1      all*      idle   2      2:1:1   3602   20462      1  (null) none
node06        1      all*      idle   2      2:1:1   3602   20462      1  (null) none
node07        1      all*      idle   2      2:1:1   3602   20462      1  (null) none
```

The `sinfo` output will show (from left-to-right):

- The hostname of your compute nodes

- The number of nodes in the list
- The node partition the node belongs to
- Current usage of the node - if no jobs are running, the state will be listed as `idle`. If a job is running, the state will be listed as `allocated`
- The detected number of CPUs (including hyper-threaded cores)
- The number of sockets, cores and threads per node
- The amount of memory in MB per node
- The amount of disk space in MB available to the `/tmp` partition per node
- The scheduler weighting

### 3.13.5 Default resources

In order to promote efficient usage of your research environment, the job-scheduler automatically sets a number of default resources for your jobs when you submit them. These defaults must be overridden by users to help the scheduler understand how you want it to run your job - if we don't include any instructions to the scheduler, then our job will take the defaults shown below:

- Number of CPU cores for your job: 1
- Number of nodes for your job: the default behavior is to allocate enough nodes to satisfy the requirements of the number of CPUs requested

You can view all default resource limits by running the following command:

```
[root@gateway1 (slurm) ~]# scontrol show config | grep Def
CpuFreqDef           = Unknown
DefMemPerNode        = UNLIMITED
MpiDefault           = none
SallocDefaultCommand = (null)
```

This documentation will explain how to change these limits to suit the jobs that you want to run. You can also disable these limits if you prefer to control resource allocation manually by yourself.

### 3.13.6 Controlling resources

In order to promote efficient usage of the research environment - the job-scheduler is automatically configured with default run-time limits for jobs. These defaults can be overridden by users to help the scheduler understand how you want it to run your job. If we don't include any instructions to the scheduler then the default limits are applied to a job.

Job instructions can be provided in two ways; they are:

1. **On the command line**, as parameters to your `sbatch` or `srun` command. For example, you can set the name of your job using the `--job-name=[name] | -J [name]` option:

```
[centos@gateway1 (scooby) ~]$ sbatch --job-name=mytestjob simplejobscript.sh
Submitted batch job 51

[centos@gateway1 (scooby) ~]$ squeue
          JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           51          all mytestjo centos  R           0:02         1 node01
```

2. **In your job script**, by including scheduler directives at the top of your job script - you can achieve the same effect as providing options with the `sbatch` or `srun` commands. Create an example job script or modify your existing script to include a scheduler directive to use a specified job name:

```
#!/bin/bash -l
#SBATCH --job-name=mytestjob
echo "Starting running on host $HOSTNAME"
sleep 120
echo "Finished running - goodbye from $HOSTNAME"
```

Including job scheduler instructions in your job-scripts is often the most convenient method of working for batch jobs - follow the guidelines below for the best experience:

- Lines in your script that include job-scheduler directives must start with `#SBATCH` at the beginning of the line
- You can have multiple lines starting with `#SBATCH` in your job-script, with normal script lines in-between
- You can put multiple instructions separated by a space on a single line starting with `#SBATCH`
- The scheduler will parse the script from top to bottom and set instructions in order; if you set the same parameter twice, the second value will be used.
- Instructions are parsed at job submission time, before the job itself has actually run. This means you can't, for example, tell the scheduler to put your job output in a directory that you create in the job-script itself - the directory will not exist when the job starts running, and your job will fail with an error.
- You can use dynamic variables in your instructions (see below)

### 3.13.7 Dynamic scheduler variables

Your research environment job scheduler automatically creates a number of pseudo environment variables which are available to your job-scripts when they are running on research environment compute nodes, along with standard Linux variables. Useful values include the following:

- `$HOME` The location of your home-directory
- `$USER` The Linux username of the submitting user
- `$HOSTNAME` The Linux hostname of the compute node running the job
- `%a / $SLURM_ARRAY_TASK_ID` Job array ID (index) number. The `%a` substitution should only be used in your job scheduler directives
- `%A / $SLURM_ARRAY_JOB_ID` Job allocation number for an array job. The `%A` substitution should only be used in your job scheduler directives
- `%j / $SLURM_JOBID` Job allocation number. The `%j` substitution should only be used in your job scheduler directives

### 3.13.8 Simple scheduler instruction examples

Here are some commonly used scheduler instructions, along with some example of their usage:

#### Setting output file location

To set the output file location for your job, use the `-o [file_name] | --output=[file_name]` option - both standard-out and standard-error from your job-script, including any output generated by applications launched by your job-script will be saved in the filename you specify.

By default, the scheduler stores data relative to your home-directory - but to avoid confusion, we recommend **specifying a full path to the filename** to be used. Although Linux can support several jobs writing to the same output file, the result is likely to be garbled - it's common practice to include something unique about the job (e.g. it's job-ID) in the output filename to make sure your job's output is clear and easy to read.

---

**Note:** The directory used to store your job output file must exist and be writable by your user **before** you submit your job to the scheduler. Your job may fail to run if the scheduler cannot create the output file in the directory requested.

---

The following example uses the `--output=[file_name]` instruction to set the output file location:

```
#!/bin/bash -l
#SBATCH --job-name=myjob --output=output.%j

echo "Starting running on host $HOSTNAME"
sleep 120
echo "Finished running - goodbye from $HOSTNAME"
```

In the above example, assuming the job was submitted as the `centos` user and was given the job-ID number 24, the scheduler will save the output data from the job in the filename `/home/centos/output.24`.

### Setting working directory for your job

By default, jobs are executed from your home-directory on the research environment (i.e. `/home/<your-user-name>`, `$HOME` or `~`). You can include `cd` commands in your job-script to change to different directories; alternatively, you can provide an instruction to the scheduler to change to a different directory to run your job. The available options are:

- `-D | --workdir=[dir_name]` - instruct the job scheduler to move into the directory specified before starting to run the job on a compute node

---

**Note:** The directory specified must exist and be accessible by the compute node in order for the job you submitted to run.

---

### Waiting for a previous job before running

You can instruct the scheduler to wait for an existing job to finish before starting to run the job you are submitting with the `-d [state:job_id] | --depend=[state:job_id]` option. For example, to wait until the job with ID 75 has finished before starting the job, you could use the following syntax:

```
[centos@gateway1 (scooby) ~]$ squeue
      JOBID PARTITION    NAME     USER ST       TIME  NODES NODELIST(REASON)
       75      all     myjob    centos R         0:01      1 node01

[centos@gateway1 (scooby) ~]$ sbatch --dependency=afterok:75 mytestjob.sh
Submitted batch job 76

[centos@gateway1 (scooby) ~]$ squeue
      JOBID PARTITION    NAME     USER ST       TIME  NODES NODELIST(REASON)
       76      all     myjob    centos PD         0:00      1 (Dependency)
       75      all     myjob    centos R         0:15      1 node01
```

## Running task array jobs

A common workload is having a large number of jobs to run which basically do the same thing, aside perhaps from having different input data. You could generate a job-script for each of them and submit it, but that's not very convenient - especially if you have many hundreds or thousands of tasks to complete. Such jobs are known as **task arrays** - an **embarrassingly parallel** job will often fit into this category.

A convenient way to run such jobs on a research environment is to use a task array, using the `-a [array_spec]` | `--array=[array_spec]` directive. Your job-script can then use the pseudo environment variables created by the scheduler to refer to data used by each task in the job. The following job-script uses the `$_SLURM_ARRAY_TASK_ID` variable to echo its current task ID to an output file:

```
#!/bin/bash -l
#SBATCH --job-name=array
#SBATCH -D $HOME/
#SBATCH --output=output.array.%A.%a
#SBATCH --array=1-1000
echo "I am $_SLURM_ARRAY_TASK_ID from job $_SLURM_ARRAY_JOB_ID"
```

```
[centos@gateway1 (scooby) ~]$ sbatch arrayjob.sh
Submitted batch job 77
[centos@gateway1 (scooby) ~]$ squeue
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
77_[85-1000]	all	array	centos	PD	0:00	1	(Resources)
77_71	all	array	centos	R	0:00	1	node03
77_72	all	array	centos	R	0:00	1	node06
77_73	all	array	centos	R	0:00	1	node03
77_74	all	array	centos	R	0:00	1	node06
77_75	all	array	centos	R	0:00	1	node07
77_76	all	array	centos	R	0:00	1	node07
77_77	all	array	centos	R	0:00	1	node05
77_78	all	array	centos	R	0:00	1	node05
77_79	all	array	centos	R	0:00	1	node02
77_80	all	array	centos	R	0:00	1	node04
77_81	all	array	centos	R	0:00	1	node01
77_82	all	array	centos	R	0:00	1	node01
77_83	all	array	centos	R	0:00	1	node02
77_84	all	array	centos	R	0:00	1	node04

All tasks in an array job are given a job ID with the format `[job_ID]_[task_number]` e.g. `77_81` would be job number `77`, array task `81`.

Array jobs can easily be cancelled using the `scancel` command - the following examples show various levels of control over an array job:

**scancel 77** Cancels all array tasks under the job ID `77`

**scancel 77\_[100-200]** Cancels array tasks `100-200` under the job ID `77`

**scancel 77\_5** Cancels array task `5` under the job ID `77`

### 3.13.9 Requesting more resources

By default, jobs are constrained to the default set of resources - users can use scheduler instructions to request more resources for their jobs. The following documentation shows how these requests can be made.

## Running multi-threaded jobs

If users want to use multiple cores on a compute node to run a multi-threaded application, they need to inform the scheduler - this allows jobs to use multiple cores without needing to rely on any interconnect. Using multiple CPU cores is achieved by specifying the `-n`, `--ntasks=<number>` option in either your submission command or the scheduler directives in your job script. The `--ntasks` option informs the scheduler of the number of cores you wish to reserve for use. If the parameter is omitted, the default `--ntasks=1` is assumed. You could specify the option `-n 4` to request 4 CPU cores for your job. Besides the number of tasks, you will need to add `--nodes=1` to your scheduler command or at the top of your job script with `#SBATCH --nodes=1`, this will set the maximum number of nodes to be used to 1 and prevent the job selecting cores from multiple nodes.

---

**Note:** If you request more cores than are available on a node in your research environment, the job will not run until a node capable of fulfilling your request becomes available. The scheduler will display the error in the output of the `squeue` command

---

## Running Parallel (MPI) jobs

If users want to run parallel jobs via a messaging passing interface (MPI), they need to inform the scheduler - this allows jobs to be efficiently spread over compute nodes to get the best possible performance. Using multiple CPU cores across multiple nodes is achieved by specifying the `-N`, `--nodes=<minnodes [-maxnodes]>` option - which requests a minimum (and optional maximum) number of nodes to allocate to the submitted job. If *only* the `minnodes` count is specified - then this is used for both the minimum *and* maximum node count for the job.

You can request multiple cores over multiple nodes using a combination of scheduler directives either in your job submission command or within your job script. Some of the following examples demonstrate how you can obtain cores across different resources;

**--nodes=2 --ntasks=16** Requests 16 cores across 2 compute nodes

**--nodes=2** Requests all available cores of 2 compute nodes

**--ntasks=16** Requests 16 cores across any available compute nodes

For example, to use 64 CPU cores on the research environment for a single application, the instruction `--ntasks=64` can be used. The following example shows launching the **Intel Message-passing MPI** benchmark across 64 cores on your research environment. This application is launched via the OpenMPI `mpirun` command - the number of threads and list of hosts are automatically assembled by the scheduler and passed to the MPI at runtime. This jobscript loads the `apps/imb` module before launching the application, which automatically loads the module for **OpenMPI**.

```
#!/bin/bash -l
#SBATCH -n 64
#SBATCH --job-name=imb
#SBATCH -D $HOME/
#SBATCH --output=imb.out.%j
module load apps/imb
mpirun --prefix $MPI_HOME \
      IMB-MPI1
```

We can then submit the IMB job script to the scheduler, which will automatically determine which nodes to use:

```
[centos@gateway1 (scooby) ~]$ sbatch imb.sh
Submitted batch job 1162
[centos@gateway1 (scooby) ~]$ squeue
      JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
      1162      all      imb      centos R        0:01      8 ip-
->10-75-1 [42,45,62,67,105,178,233,250]
```

(continues on next page)

(continued from previous page)

```
[centos@gateway1 (scooby) ~]$ cat imb.out.1162
#-----
# Intel (R) MPI Benchmarks 4.0, MPI-1 part
#-----
# Date           : Tue Aug 30 10:34:08 2016
# Machine        : x86_64
# System         : Linux
# Release        : 3.10.0-327.28.3.el7.x86_64
# Version        : #1 SMP Thu Aug 18 19:05:49 UTC 2016
# MPI Version    : 3.0
# MPI Thread Environment:

#-----
# Benchmarking PingPong
# #processes = 2
# ( 62 additional processes waiting in MPI_Barrier)
#-----
#bytes #repetitions      t[usec]  Mbytes/sec
      0         1000         3.17         0.00
      1         1000         3.20         0.30
      2         1000         3.18         0.60
      4         1000         3.19         1.19
      8         1000         3.26         2.34
     16         1000         3.22         4.74
     32         1000         3.22         9.47
     64         1000         3.21        19.04
    128         1000         3.22        37.92
    256         1000         3.30        73.90
    512         1000         3.41       143.15
   1024         1000         3.55       275.36
   2048         1000         3.75       521.04
   4096         1000        10.09       387.14
   8192         1000        11.12       702.51
  16384         1000        12.06      1296.04
  32768         1000        14.65      2133.32
  65536          640        19.30      3238.72
 131072          320        29.50      4236.83
 262144          160        48.17      5189.77
 524288           80        84.36      5926.88
1048576           40       157.40      6353.32
2097152           20       305.00      6557.31
4194304           10       675.20      5924.16
```

**Note:** If you request more CPU cores than your research environment can accommodate, your job will wait in the queue. If you are using the Flight Compute auto-scaling feature, your job will start to run once enough new nodes have been launched.

### 3.13.10 Requesting more memory

In order to promote best use of the research environment scheduler - particularly in a shared environment, it is recommended to inform the scheduler the maximum required memory per submitted job. This helps the scheduler appropriately place jobs on the available nodes in the research environment.

You can specify the maximum amount of memory required per submitted job with the `--mem=<MB>` option. This

informs the scheduler of the memory required for the submitted job. Optionally - you can also request an amount of memory *per CPU core* rather than a total amount of memory required per job. To specify an amount of memory to allocate *per core*, use the `--mem-per-cpu=<MB>` option.

**Note:** When running a job across multiple compute hosts, the `--mem=<MB>` option informs the scheduler of the required memory *per node*

### 3.13.11 Requesting a longer runtime

In order to promote best-use of the research environment scheduler, particularly in a shared environment, it is recommend to inform the scheduler the amount of time the submitted job is expected to take. You can inform the research environment scheduler of the expected runtime using the `-t, --time=<time>` option. For example - to submit a job that runs for 2 hours, the following example job script could be used:

```
#!/bin/bash -l
#SBATCH --job-name=sleep
#SBATCH -D $HOME/
#SBATCH --time=0-2:00
sleep 7200
```

You can then see any time limits assigned to running jobs using the command `squeue --long`:

```
[centos@gateway1 (scooby) ~]$ squeue --long
Tue Aug 30 10:55:55 2016
          JOBID PARTITION      NAME      USER      STATE      TIME TIME_LIMI  NODES_
↪NODELIST (REASON)
          1163      all      sleep      centos  RUNNING      0:07   2:00:00     1_
↪ip-10-75-1-42
```

### 3.13.12 Further documentation

This guide is a quick overview of some of the many available options of the SLURM research environment scheduler. For more information on the available options, you may wish to reference some of the following available documentation for the demonstrated SLURM commands;

- Use the `man squeue` command to see a full list of scheduler queue instructions
- Use the `man sbatch/srun` command to see a full list of scheduler submission instructions
- Online documentation for the SLURM scheduler is [available here](#)